计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330127
60(5): 964−973, 2023

机器学习方法赋能系统软件:挑战、实践与展望

唐楚哲　王肇国　陈海波
(上海交通大学软件学院　上海　200240)
(领域操作系统教育部工程研究中心(上海交通大学)上海　200240)
(t.chuzhe@sjtu.edu.cn)

Empowering System Software with Machine Learning Methods: Challenges, Practice, and Prospects

Tang Chuzhe, Wang Zhaoguo, and Chen Haibo
(*School of Software*, *Shanghai Jiao Tong University*, *Shanghai* 200240)
(*Engineering Research Center for Domain-specific Operating Systems* (*Shanghai Jiao Tong University*), *Ministry of Education*, *Shanghai* 200240)

Abstract　Machine learning methods have brought new opportunities for building system software that fully utilizes hardware resources to support emerging applications. However, in order to adapt to the demands of various application scenarios, system software design and implementation need continuous improvement and evolution. Meanwhile, machine learning methods have the potential to extract patterns from data and automatically optimize system performance. Despite this potential, applying machine learning methods to empower system software faces several challenges, such as customizing models for system software, obtaining training data with sufficient quality and quantity, reducing the impact of model execution costs on system performance, and avoiding the hindrance of model errors on system correctness. We present the practical experience of the Institute of Parallel and Distributed Systems (IPADS) at Shanghai Jiao Tong University in applying machine learning methods to optimize system software for index structures, key-value storage systems, and concurrency control protocols. The lessons learned from the practice in model design, system integration, and practitioner knowledge are summarized. Additionally, we briefly review relevant research at home and abroad, and propose prospects and suggestions for this line of research, including collaboration between systems and machine learning experts, building modular, reusable system prototypes, and exploring model optimization techniques dedicated to systems context. The aim is to offer references and help for future work.

Key words　machine learning; system software; index structure; key-value store; concurrency control

摘　要　机器学习方法为构建系统软件带来了新的机遇.为充分利用硬件资源支撑新型应用,系统软件的设计与实现需要不断改进与演化,以适应不同场景的需求.机器学习方法具有从数据中提取规律并自动优化系统性能的潜力.然而,使用机器学习方法赋能系统软件面临一些挑战,包括设计面向系统软件的定制化模型、获取足量且高质量的训练数据、降低模型开销对系统性能的影响,以及消除模型误差对系统正确性的影响等.介绍了上海交通大学并行与分布式系统研究所在索引结构、键值存储系统、并发控制协议等方面应用机器学习方法优化系统软件的实践,并从模型设计、系统集成和实践者自身知识储备等方面

收稿日期:2023-02-28;**修回日期**:2023-03-30
基金项目:国家自然科学基金项目(61925206, 62272304, 62132014)
This work was supported by the National Natural Science Foundation of China (61925206, 62272304, 62132014).
通信作者:陈海波(haibochen@sjtu.edu.cn)

总结了经验与教训. 此外，还回顾了国内外相关研究，并对此研究方向提出了展望与建议，希望为未来的研究提供参考与帮助.

关键词 机器学习；系统软件；索引结构；键值存储；并发控制

中图法分类号 TP31

随着信息技术的发展，操作系统、数据库系统等系统软件已广泛应用于医疗、工业、军事、航空航天和能源等关键领域以及人们的日常生活中，是现代社会不可或缺的基础设施. 同时，伴随着人工智能、大数据、云计算等新技术的进步，以及可编程存储器、RDMA（remote direct memory access）网络硬件、可编程交换机等新硬件的涌现，为充分利用硬件支撑上层应用，系统软件的设计与实现不断面临着与时俱进的新要求. 这不仅是对系统软件实践者的挑战，也为系统软件研究带来新的机遇.

与此同时，机器学习方法近年来取得了令人瞩目的进展. 2015 年，微软的研究人员使用深度残差网络（deep residual network, ResNet）在 ImageNet 测试集中达到超越人类表现[1]；2016 年，DeepMind 的研究人员使用深度强化学习（deep reinforcement learning, DRL）训练出 AlphaGo 模型并击败了围棋世界冠军李世石[2-3]；2018—2020 年，OpenAI 的研究人员基于 Transformer 模型架构[4] 提出 GPT 系列模型[5-7]，并于 2022 年向公众发布 ChatGPT 模型，ChatGPT 凭借其优秀的对话与推理能力引起社会各界广泛的关注与讨论.

面对构建复杂系统软件的迫切需求和机器学习方法的快速发展，一些系统研究人员开始思考并尝试回答：能否使用机器学习方法来赋能系统软件？近年来，研究领域中出现不少应用机器学习方法于系统软件的尝试，并受到研究人员与从业者的共同关注. 值此之际，本文将讨论使用机器学习方法赋能系统软件的机遇与挑战，介绍将机器学习方法应用于索引结构、键值存储系统、并发控制协议等方面的实践，总结经验与教训，并回顾国内外相关研究，希望为尝试应用机器学习方法的系统软件研究人员与从业者提供参考与帮助.

1 机器学习方法赋能系统软件的机遇与挑战

随着软硬件环境的不断演变，系统软件的设计与实现日益复杂. 图灵奖得主 Lampson[8-9] 曾指出，系统软件不仅需要充分利用底层硬件资源为上层应用提供及时与高效的服务，同时还需要保证系统的稳定性和可靠性，并能够快速适应需求与环境的变化. 这些要求为机器学习方法在系统软件中发挥作用带来了新的机遇. 例如，可以使用机器学习方法来优化系统资源的利用和分配，以及实现自适应的系统配置和调整，从而提高系统软件的性能和可靠性. 同时，机器学习方法也可以对系统软件产生的大量运行数据进行分析和挖掘，以优化系统软件的运行效率和性能，从而满足不断增长的应用需求.

然而，机器学习方法赋能系统软件面临 4 点挑战：

1）面临设计面向系统软件定制化模型的挑战. 系统软件往往有复杂的架构与多样化的功能，以及与常见的机器学习任务所不同的优化目标，因此需要为具体系统和应用场景设计定制化的机器学习模型，以有效地进行优化. 因此，系统研究人员需要深入了解系统软件的工作原理和性能瓶颈，并具备机器学习领域的专业知识和经验，才能设计出合适的模型.

2）面临获取足量且高质量训练数据的挑战. 系统软件所使用的模型训练数据往往需要从系统运行过程中收集而来，但系统的每次运行往往会消耗大量时间与计算资源，而只能产生少量的训练数据，这给获取训练数据带来了成本和效率方面的挑战. 此外，为满足系统软件对稳定性与可靠性的高要求，训练数据往往需要覆盖各种可能的工作负载和异常情况，从而保证训练出的模型具有较好的鲁棒性和泛化能力，这进一步增加训练数据的获取难度.

3）面临降低模型开销对系统性能影响的挑战. 与数据分析等其他应用机器学习方法的场景不同，系统软件对实时性和并发度有极高的要求，因此对于模型执行的效率和响应时间要求非常高. 然而，由于神经网络等机器学习模型通常需要大量的计算资源和时间来执行，这会导致模型的执行开销影响系统软件的性能. 尤其是在嵌入式设备和边缘设备等资源受限的环境中，模型执行开销甚至可能大于使用机器学习方法带来的提升，进而对系统软件造成负面影响.

4）面临消除模型误差对系统正确性影响的挑战. 对于系统软件而言，始终保证系统行为的正确性至

关重要. 然而, 机器学习模型产生的结果总是伴随着误差, 难以提供精确度保证. 因此, 如何在使用模型输出时避免产生错误的系统行为也是应用机器学习方法所面临的重大挑战.

2 机器学习方法赋能系统软件的实践

在系统软件实践者的视角下, 机器学习方法可以被简单地看作一种函数近似器[10]. 因此, 任何能够被表示为某种函数形式的系统组件都可以被机器学习模型近似拟合. 以数据库查询优化器为例, 它的功能可以总结为将输入查询语句进行优化并输出具体的查询执行计划. 因此, 一个以查询语句抽象语法树(abstract syntax tree, AST)和数据库内部统计信息为输入, 输出物理执行计划的函数就可以表达查询优化器这一系统组件. 通过收集系统运行时查询优化器的真实输入输出以构建训练数据集, 这样就使得采用机器学习方法替换这一组件成为可能. 尽管模型会产生误差, 但这迈出了用机器学习方法优化关键系统软件组件的第一步.

下一个问题随之而来: 哪些函数适合被机器学习方法所近似拟合呢? 本节通过介绍上海交通大学并行与分布式系统研究所及相关合作者将机器学习方法应用于系统软件的 3 个具体案例来探讨这个问题. 本文旨在抛砖引玉, 而非给出一个通用的答案.

2.1 机器学习方法在索引结构上的实践

索引结构是包括存储系统在内的许多系统软件中重要的组成部分, 它们可以显著提高数据检索的速度和效率, 减少系统资源的消耗, 从而提升整个系统的性能. 常见的索引结构包括 B+树、哈希表及其变种等. 索引结构本质上是一类记录数据存储位置并提供查找能力的数据结构, 因此可以用一个以查询键(key)为输入, 输出数据位置的函数来表示其功能. 因此, 如图 1 所示, 用机器学习方法替代传统索引结构的核心在于使用机器学习模型来近似拟合这一函数.

使用机器学习方法近似拟合索引结构函数存在 3 个难点: 1)机器学习模型必须具有高效的执行速度. 因为访问索引结构是存储系统查询处理关键路径的组成部分, 机器学习模型推理时间必须是微秒级别才能避免对系统性能造成负面影响. 2)使用模型后, 查询结果必须始终准确. 用户期望索引结构查询时返回且只返回匹配查询键数据的存储位置, 遗漏数据或返回错误数据位置都会严重影响系统的可

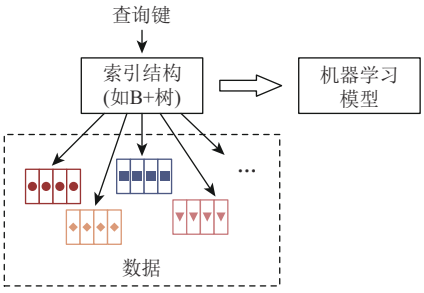


Fig. 1 Illustration of replacing index structures with machine learning models

图 1 机器学习模型代替索引结构的示意图

靠性. 3)模型必须能够快速适应数据更新. 插入、删除等操作会改变索引结构对查询键的输出, 进而改变模型所近似拟合的函数. 如果不能及时更新模型以适应更新后的函数, 系统正确性仍无法得到保障.

对于第 1 个难点, 首先对导致模型执行效率低的原因进行分析. 我们发现, 模型执行效率低往往是由于使用深度神经网络(deep neural network, DNN)等复杂模型导致, 而之所以需要使用复杂模型则是因为所需近似拟合的函数本身较为复杂, 需要高容量(capacity)的模型才能够提供足够的精度. 因此, 解决模型执行效率问题的核心在于简化所需近似拟合函数的复杂度. 如图 2 所示, 索引函数复杂是因为被索引数据往往是随机分散在存储介质的不同位置上, 从而导致索引结构所代表的查询键与数据位置间的映射缺乏规律. 因此, 只需要将数据进行排序, 并使其连续存放于存储介质中, 那么模型所需近似拟合的函数将单调递增, 从而使用简单的线性模型就可以较好地对其进行近似. 这一思想最初由 Kraska 等

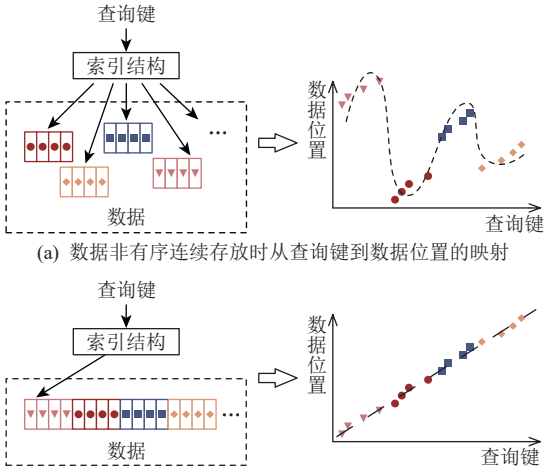


Fig. 2 Sorting data and storing date continuously simplify the approximated function

图 2 将数据排序并连续存放可简化被拟合函数

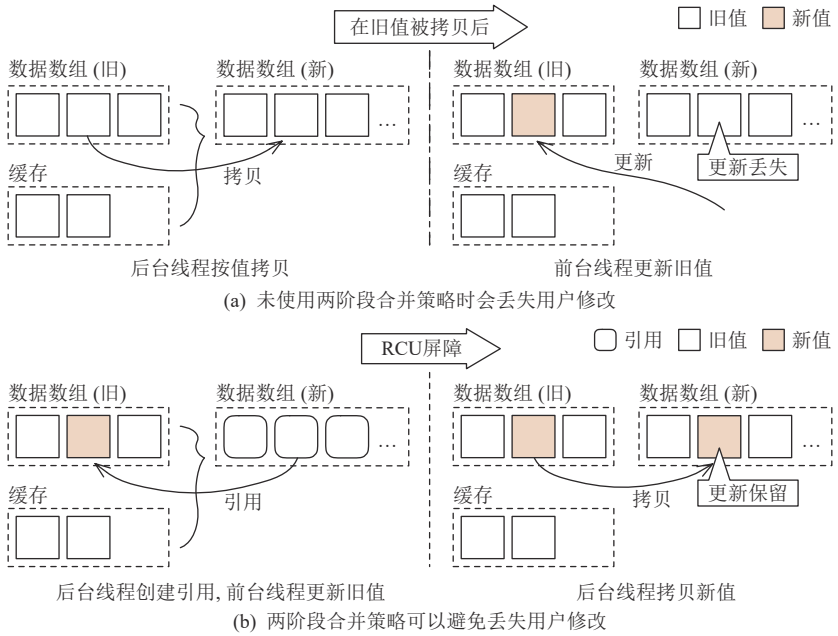
人[11]提出，并进一步构建了递归模型索引（recursive model index, RMI）架构来索引数据.

对于第 2 个难点，需要为模型误差设计一个后备机制（fallback mechanism）. 具体而言，尽管模型预测的数据位置可能存在误差，但通常情况下预测位置与数据真实位置差距并不大. 考虑到数据本身已经被排序并连续地存放于存储介质中，可以从预测位置出发进行指数搜索（exponential search），通过将当前位置数据与用户给定查询键进行比较，最终定位到数据真实位置. 指数搜索的复杂度为 $O(\log(n))$，其中 n 为模型预测误差. 因此，有了这样的后备机制，只需尽可能提高模型精度以减少指数搜索开销，而不用担心输出错误的查询结果.

对于第 3 个难点，发现直观的数据更新适应方案要么严重影响系统性能，要么会导致索引结构数据一致性问题. 为了简化索引函数并使用指数搜索来确定数据真实位置，被索引数据需要排序并连续地存储，因此每次数据插入与删除将会涉及到大量数据的移动操作. 就算使用内存作为存储介质，这一过程仍可消耗数秒时间来完成. 同时，数据位置发生改变时，模型也需要及时更新，以保证查询误差始终处于较低状态，这进一步加大了应对数据更新的开销. 对于这一问题，直观的解决方法是为数据修改设计单独的缓存，缓存足够大时在后台异步地完成数据移动与模型更新操作. 尽管这些操作耗费的时间未减少，但它们不再阻塞正常的索引查询与更新.

如图 3 所示，当后台线程将数据从旧索引的数据数组和缓存拷贝至新索引的数据数组时，前台线程仍然可以对旧索引进行更新操作. 由于后台线程此时仍未完成新索引的构建，新索引的数据数组无法暴露给前台线程，因此当前台线程对已被拷贝至新索引的数据进行更新时，拷贝后的数据并不会被正确更新. 这会导致数据一致性问题，即在后台线程完成新索引构建后，旧索引将被回收，导致用户成功完成的更新消失. 为了解决这个问题，我们提出了两阶段压缩（two-phase compaction）方法来正确进行新索引构建. 简单来说，将后台线程构建新索引的过程分为 2 个阶段：归并（merge）阶段和拷贝（copy）阶段. 在归并阶段，后台线程将旧索引的数据数组和缓存中的数据按照查询键进行归并排序，并将指向它们的指针存入新索引的数据数组中. 完成归并阶段后，我们使用 RCU（read-copy-update）机制将所有前台线程的索引视图迁移至新构建的索引，进入拷贝阶段将新索引的数据数组中的指针替换为其所引用的具体数据. 使用两阶段压缩技术后，只有当所有前台线程均停止通过旧索引修改数据，才会执行数据拷贝，从而避免了直接拷贝数据后被前台线程修改导致的数据不一致问题.

如图 4 所示，我们的测试表明，将机器学习方法应用于索引结构来构建的 XIndex 索引结构[12-14]，不仅拥有超越传统索引结构的查询性能，而且在读写负载下具有优秀的可扩展性.



(a) 未使用两阶段合并策略时会丢失用户修改

(b) 两阶段合并策略可以避免丢失用户修改

Fig. 3 Updating indexes with two-phase compaction avoids data consistency issue

图 3 通过两阶段压缩更新索引可避免数据一致性问题

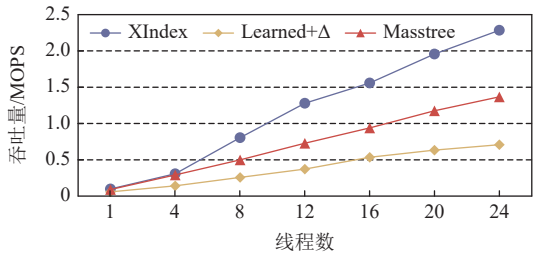


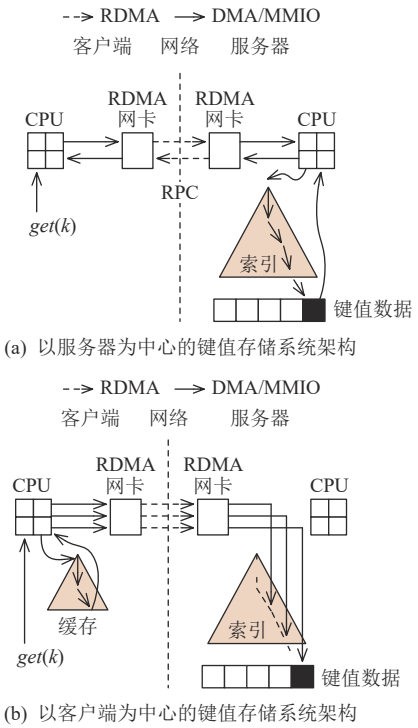
Fig. 4 XIndex achieves good scalability in a TPC-C style workload and outperforms state-of-the-art baselines

图 4 在类 TPC-C 负载下 XIndex 可扩展性良好且性能优于现有系统

2.2 机器学习方法在键值存储系统上的实践

键值存储系统是一类重要的系统软件，它们能够快速地存储和检索大量的键值对数据. 许多大型分布式系统(如分布式数据库)均使用键值存储系统作为存储模块并在其上构建更为复杂的功能.

对于键值存储系统，数据索引效率是决定其性能的关键. 如图 5 所示，传统客户端服务器模式的键值存储系统中，客户端与服务器以远程过程调用(remote procedure call, RPC)的方式进行通信. 客户端向服务器发送的读写请求需经由服务器端 CPU 进行处理后再将结果返回给客户端. 随着近年来 RDMA 技术的发展，客户端与服务器间网络通信能力不断提升[15]. 然而服务器端 CPU 处理能力却未能获得相匹配的提高，因此这类架构的键值存储系统中服务器端 CPU 处理请求的性能逐渐成为整个系统的瓶颈. 与此同时，RDMA 技术允许客户端直接对服务器端内存进行访问而不需服务器端 CPU 的参与，这使另一种直接由客户端驱动数据查询的架构成为可能. 在这种架构下客户端将代替原服务器端 CPU 完成对服务器端所管理的数据进行访问. 然而，服务器端数据往往以树状结构进行组织，以提供按顺序扫描的功能. 若简单地让客户端复刻服务器端 CPU 进行的查询过程，那对树状数据结构每一层的访问都将需要一次网络往返(network roundtrip). 这不仅会大大增加查询操作所需的时间，还会加剧对网络带宽资源的消耗.



(a) 以服务器为中心的键值存储系统架构

(b) 以客户端为中心的键值存储系统架构

Fig. 5 Architectures of RDMA-based key-value stores

图 5 RDMA 键值存储系统架构

怎样设计键值存储系统才能充分发挥 RDMA 技术带来的卓越硬件性能成为我们思考的一个问题. 我们发现，在客户端对服务器端索引结构进行缓存可以有效减少查询时的网络往返，但这种方法会对客户端造成极大的内存空间开销，并在读写负载下导致大量缓存失效(cache invalidation)和缓存未命中(cache miss)，引起系统性能下降.

为此，我们提出使用机器学习方法替代客户端的索引结构缓存，并称其为学习缓存(learned cache). 正如 2.1 节所提到，机器学习方法可以有效替代传统索引结构，通过模型计算来替代传统索引结构中逐层访问过程，在提供高效查询性能的同时大幅度减少索引结构所需内存空间. 实现这一想法最大的难点来自于真实负载下数据的动态变化. 我们在 XIndex 中使用缓存来应对删除和插入操作，但这种思想难以用于 RDMA 键值存储系统. 首先，服务器端引入额外的缓存结构会导致客户端查询时额外的网络往返，从而严重增加查询延迟. 其次，对写操作有较好支持的缓存结构通常也是树状的，很难在客户端缓存这样快速变化的缓存结构. 最后，对数据和模型的重构会中断客户端发起的 RDMA 远程访问，并导致客户端的学习缓存完全失效.

为解决这些问题，我们提出了一种混合的系统架构并构建了原型系统 XStore[16,17]. 如图 6 所示，该架构保留了服务器端的 B+树索引结构以处理动态工作负载，并在客户端上使用学习缓存来处理静态工作负载. 对于 *get*(), *scan*()等只读请求，客户端首先使用学习缓存预测查询键的位置范围，然后通过一次 RDMA 读操作获取这个范围的数据. 最后，客户端在本地获取的数据范围内搜索找到数据的实际位置，并使用另一次 RDMA 读操作获取完整数据. 对于 *update*(), *insert*(), *delete*()等写操作，客户端将使用

传统 RPC 的方式将请求发给服务器. 服务器首先通过遍历 B+树索引确定查询键的位置，然后插入新的键值对. 系统将在后台重新训练被更新树节点所对应的模型，客户端会根据需求独立地从服务器端获取新的学习缓存.

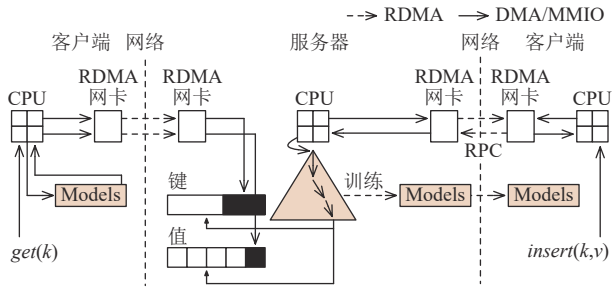


Fig. 6 The architecture of XStore

图 6 XStore 系统架构

如图 7 所示，测试表明 XStore 能够在只读负载下每秒处理 8 000 万次请求，在读写负载下每秒处理 5 300 万次请求，相比于当前最先进的 RDMA 键值存储系统分别实现了高达 5.9 倍和 3.5 倍的提升.

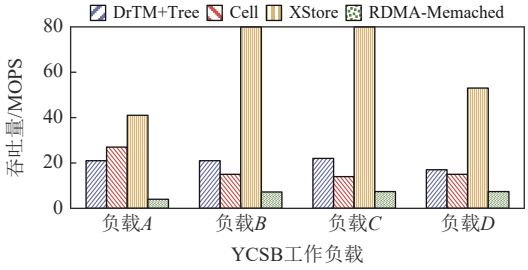


Fig. 7 XStore outperforms state-of-the-art baselines in both read-only workload and read-write workload

图 7 在只读和读写负载下 XStore 性能均优于现有系统

2.3 机器学习方法在并发控制协议上的实践

并发控制协议是并发系统高效执行用户请求并保证执行结果正确的关键组件. 在数据库系统中并发控制协议是保证用户事务隔离性(isolation)的核心，它通过协调并发事务中各个操作的执行，来保证高效的事务执行与正确的事务语义，例如可串行化(serializability). 目前，为了最大化并发系统的性能，研究人员通常会针对具体应用负载场景手动设计并发控制协议. 但是，当工作负载发生变化时，原有的针对性并发控制协议优化将难以发挥作用，甚至低于其他简单的经典算法.

能否利用机器学习方法的适应能力来构建能够适配各种工作负载的并发控制协议是我们思考的一个问题. 首先，需要寻找能够表示并发控制协议的函数抽象. 受到强化学习的启发，我们发现尽管目前存在多种风格的并发控制协议，但它们都可以被简单地视为一个以事务当前执行状态为输入，并输出这个事务下一个行动(action)的函数. 这为我们应用机器学习方法构建并发控制协议奠定了基础.

使用机器学习方法近似拟合这个函数存在 3 个难点：1)如何表示事务的执行状态. 选择合适的表示方式需要考虑模型的执行开销、收集训练数据的日志开销，以及不同表示方式所对应的状态空间大小等. 因此需要一个足够精炼的表示方式以避免潜在的性能问题. 2)如何定义并表示事务的行动. 类似于执行状态表示方式，事务行动的定义与表示需要足够精炼. 此外，事务行动的定义与表示还需要覆盖现有并发控制协议所提供的行动，以保证我们学习的并发控制协议能够拟合现有算法. 3)如何保证近似拟合得到的并发控制协议的正确性. 例如保证被协调的事务执行结果可串行化.

对于第 1 个难点，我们希望状态表示方式能够区分出需要不同行动的事务状态. 经过探索，发现使用由事务类型和操作 ID 组成的元组是一个足够简单并能够精确区分事务执行所需执行的不同操作的表达方式. 例如，在 TPC-C 负载中，可以通过 New-Order 事务类型及其第 2 个操作来表示当前事务执行状态.

对于第 2 个难点，我们希望行动的定义和表示能够足够细粒度地控制并发事务的交错执行，并且能够对大部分现有并发控制协议进行编码. 经过分析发现，行动定义应包括以下决定：是否需要等待以及等待多久、选取数据的哪个版本来完成读操作、是否将未提交的脏写(dirty write)变得可见、立即对事务执行进行验证还是在提交前再验证等. 最终，将这些可能的行动以类似独热编码(one-hot encoding)的方式构建了行动空间.

至此，我们将学习的并发控制协议表示为由事务状态和行动构成的二维表格形式(见图 8)，使用遗传算法在不同工作负载下学习适配当前负载的并发控制策略(见图 9)，并构建了原型系统 Polyjuice[18].

对于第 3 个难点，我们使用数据库系统中常见的事务执行验证机制作为机器学习并发控制协议的后备机制，从而保证无论模型产生了怎样的并发控制策略，都只有在符合用户指定事务语义的执行结果时才能提交. 具体而言，我们使用了一个类似于 Silo 数据库的验证机制[19].

经测试，我们发现 Polyjuice 能在不同的工作负载下实现高于针对这些负载进行优化的并发控制协议的性能，如图 10 所示. 仔细观察 Polyjuice 的事务协

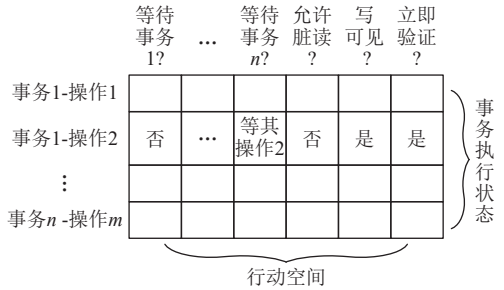


Fig. 8 Concurrency control policy table representation used in Polyjuice

图 8 Polyjuice 使用的并发控制策略表格表示方式

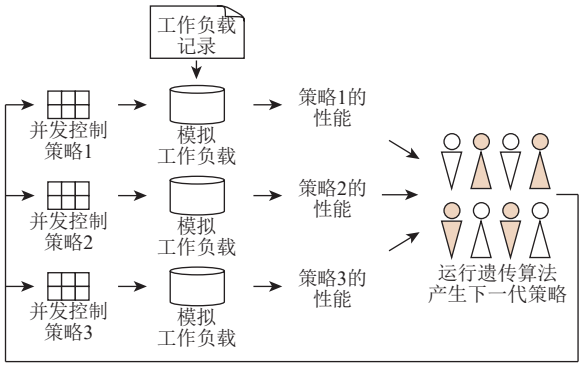


Fig. 9 Polyjuice uses the evolutionary algorithm to train concurrency control policies for specific workloads

图 9 Polyjuice 用遗传算法为特定工作负载训练并发控制策略

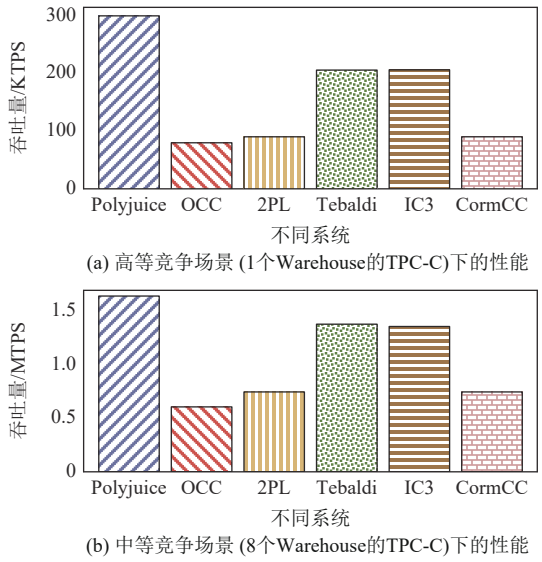


Fig. 10 Polyjuice outperforms existing concurrency control protocols under different workloads

图 10 不同工作负载下 Polyjuice 都能优于现有并发控制协议

调策略，我们发现通过遗传算法学习，Polyjuice 能够输出现有并发控制协议无法做到的更高效的事务交错执行方式，如图 11 所示. 这进一步证明了机器学习方法在为系统软件探索策略空间方面的巨大潜力.

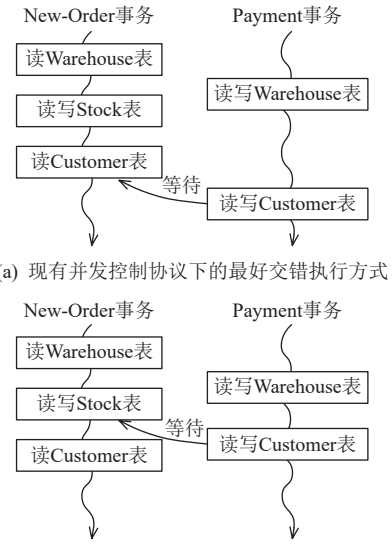


Fig. 11 Polyjuice can interleave transactions more efficiently in the TPC-C workload

图 11 Polyjuice 能更高效地交错执行 TPC-C 负载的事务

3 机器学习方法赋能系统软件的经验与教训

将机器学习方法应用到系统软件的设计与实现中存在诸多挑战. 本节将介绍在实践中应对这些挑战的经验与教训，希望能对未来机器学习方法赋能系统软件起到借鉴作用.

首先，我们认为部署在系统软件中的机器学习模型应该尽可能地简单. 模型执行往往是系统关键路径的一部分，使用简单模型可以有效避免机器学习方法引入过多额外开销. 为了使简单模型有限的近似能力在系统软件中发挥作用，我们需要使用简单的问题表述(problem formulation)，从而使简单模型成为可选项. 以机器学习数据索引为例，我们曾尝试使用神经网络模型来替代线性模型，但神经网络模型不仅收敛过程存在不确定性，而且它们的训练和推断时间相比于线性模型高了 1 个数量级，无法用于数据索引. 以机器学习并发控制为例，我们曾尝试使用深度强化学习方法来学习并发控制策略. 然而，强化学习智能体(agent)的收敛过程同样具有极大不确定性，并且经常收敛于非最优点，其性能低于遗传算法学得的策略. 因为使用真实数据库系统运行工作负载来获得强化学习所需要的环境反馈开销巨大，难以收集更多训练数据来进一步探索强化学习方法的可行性，而是转而使用遗传算法.

其次，我们认为应用机器学习方法时总是需要设计相应的后备机制. 机器学习模型无法保证完全精确，也难以对其误差范围提供确切保证，而系统软件往往需要提供正确性保证，特别是对查询优化器这类本身解决优化问题的系统组件而言更是如此. 因此，我们认为将不精确的机器学习模型与纠正误差的后备机制相结合是机器学习方法赋能系统软件的一种潜在范式. 以机器学习数据索引为例，为解决模型输出的数据位置误差，使用了指数搜索方法来探寻数据的正确位置. 这一后备机制的开销与模型误差成正比，因此当模型精度足够高时，并不需要为后备机制付出额外开销. 以机器学习并发控制为例，我们复用了 Silo 中为乐观并发控制（optimistic concurrency control, OCC）设计的验证方法来保证提交事务的正确性. 事务验证机制原本就是许多并发控制协议所必需的一部分，仅对 Silo 验证机制进行极少修改，从而在保证所学并发控制策略正确性的同时避免引入大量开销.

最后，我们认为系统软件实践者应在引入机器学习方法的同时继续加深对目标系统的理解. 对系统功能进行建模并非易事，它不比模型调优简单，而往往需要我们对系统有了深入理解后才能凝练出合适的学习任务. 因此，当我们发现模型难以达到预期效果时，不妨回过头审视自身对系统的理解以及对问题的表述. 以机器学习数据索引为例，我们通过将数据排序并连续存储以简化所需近似拟合的索引函数，进而使用线性模型完成拟合. 以机器学习并发控制为例，我们将大部分精力耗费在探寻简单并具备足够表达能力的事务状态与行动编码方法，而非尝试不同的模型架构与训练方法. 只有深入理解和分析并发控制协议优化的相关研究，我们才能获得最终有效的编码方式，应用机器学习方法才能有所成效.

4 国内外相关研究

机器学习方法赋能系统软件是近期新兴的研究方向，国内外研究人员均开展了相关工作. 本节对国内外相关研究进行简要回顾.

自 2018 年麻省理工学院和谷歌的研究人员提出学习索引结构（learned index structure）[11] 以来，机器学习方法赋能系统软件这一研究方向已引起了系统研究领域的广泛关注. 麻省理工学院的研究人员随后在数据查询执行[20-21] 和查询优化[22-23] 方面进一步尝试应用机器学习方法，并提出 SageDB 数据库学术原型，将多种基于机器学习方法的数据库组件结合来实现数据分布、工作负载和硬件环境自适应[24-25]. 在国内方面，上海交通大学的研究人员提出了支持动态负载的高可扩展学习索引结构[12-14]，华中科技大学的研究人员则进一步改进了学习索引结构的可扩展性[26].

学习索引结构的提出进一步启发了机器学习方法在存储系统中的应用. 威斯康星大学和微软的研究人员提出使用学习索引结构优化日志结构合并树（log-structured merge-tree, LSM tree）的查询性能[27]. 在国内方面，上海交通大学的研究人员提出了一种基于学习缓存与传统 B+树混合架构的 RDMA 键值存储系统[16-17]，华中科技大学的研究人员则基于学习索引结构构建了面向分离内存（disaggregated memory）场景下的 RDMA 键值存储系统[28]. 此外，针对机器学习方法在数据库系统中的应用，中国人民大学的研究人员开展了系统性调研并指出数个重要研究问题与潜在挑战[29].

5 总结与展望

机器学习方法为系统软件的优化和改进提供了新的思路与手段. 实践表明，机器学习方法在优化关键系统组件方面极具潜力. 然而，应用机器学习方法并非易事，实践者将面临定制学习任务与模型、获取训练数据、处理模型开销和误差等方面的新挑战. 回顾在这一方向上的实践经验，我们发现简单地将现有方法应用到系统组件中的效果往往不尽如人意，我们不仅需要同时具备系统和机器学习知识，还需要不断调整系统设计和机器学习算法. 为了应对这些挑战，我们从模型设计、系统集成和实践者自身知识储备等方面总结了经验与教训.

目前，机器学习方法赋能系统软件仍是一个具有较高门槛的研究方向. 一方面，系统研究本身需要大量工程开发并依赖过往经验积累；另一方面，从系统软件实践者的角度来看，机器学习方法更像黑盒工具，难以明确该如何恰当地应用它们. 为了促进这一方向的研究，我们建议：

1）建立系统研究人员和机器学习专家之间的协作关系，共同探索如何用机器学习方法赋能系统软件.

2）以可组合可重用的思想构建系统原型，以便于降低引入与分析机器学习模型难度，更有利于基于过往研究成果与系统代码开展新的研究.

3)探索面向系统软件的模型优化技术,以获得更加适配系统软件运行限制的模型,包括但不限于缩短训练时间、减小内存占用和保证模型精度等方面.

机器学习方法赋能系统软件的研究方兴未艾.随着技术的不断进步和应用场景的不断扩展,我们相信机器学习方法在系统软件方面的应用前景将会更加广阔,在实际场景中发挥越来越大的作用.

作者贡献声明: 唐楚哲负责文献调研和整理,并撰写前言与第2节和第4节;王肇国负责撰写论文其余部分;陈海波负责确立论文框架、提出指导意见并修改论文.

参 考 文 献

[1] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep residual learning for image recognition [C] //Proc of the 2016 IEEE Conf on Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2016: 770−778

[2] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge[J]. Nature, 2017, 550(7676): 354−359

[3] Silver D, Huang A, Maddison C, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484−489

[4] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need [C] //Advances in Neural Information Processing Systems 30. Red Hook, NY: Curran Associates Inc. , 2017: 6000−6010

[5] Radford A, Narasimhan K, Salimans, T, et al. Improving language understanding by generative pre-training [R/OL]. San Francisco, CA: OpenAI, 2018 [2023-02-27].https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[6] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners [R/OL]. San Francisco, CA: OpenAI, 2019 [2023-02-27].https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[7] Brown T B, Mann B, Ryder N, et al. Language models are few-shot learners [C] //Advances in Neural Information Processing Systems 33. Red Hook, NY: Curran Associates Inc. , 2020: 1877−1901

[8] Lampson B W. Hints for computer system design [C] //Proc of the 9th ACM Symp on Operating Systems Principles. New York: ACM, 1983: 33−48

[9] Lampson B W. Hints and principles for computer system design [R/OL]. Ithaca, NY: CoRR, 2020 [2023-02-27].https://arxiv.org/abs/2011.02455

[10] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators[J]. Neural Networks, 1989, 2(5): 359−366

[11] Kraska T, Beutel A, Chi E H, et al. The case for learned index structures [C] //Proc of the 2018 Int Conf on Management of Data. New York: ACM, 2018: 489−504

[12] Tang Chuzhe, Wang Youyun, Dong Zhiyuan, et al. XIndex: A scalable learned index for multicore data storage [C] //Proc of the 25th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2020: 308−320

[13] Wang Youyun, Tang Chuzhe, Wang Zhaoguo, et al. SIndex: A scalable learned index for string keys [C] //Proc of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems. New York: ACM, 2020: 17−24

[14] Wang Zhaoguo, Chen Haibo, Wang Youyun, et al. The concurrent learned indexes for multicore data storage[J]. ACM Transactions on Storage, 2022, 18(1): 1−35

[15] Chen Youmin, Lu Youyou, Luo Shengmei, et al. Survey on RDMA-based distributed storage systems[J]. Journal of Computer Research and Development, 2019, 56(2): 227−239 (in Chinese)
(陈游旻, 陆游游, 罗圣美, 等. 基于RDMA的分布式存储系统研究综述[J]. 计算机研究与发展, 2019, 56(2): 227−239)

[16] Wei Xingda, Chen Rong, Chen Haibo, et al. 2021. XStore: Fast RDMA-based ordered key-value store using remote learned cache[J]. ACM Transactions on Storage, 2021, 17(3): 1−32

[17] Wei Xingda, Chen Rong, Chen Haibo. Fast RDMA-based ordered key-value store using remote learned cache [C] //Proc of the 14th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 117−135

[18] Wang Jiachen, Ding Ding, Wang Huan, et al. Polyjuice: High-performance transactions via learned concurrency control [C] //Proc of the 15th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2021: 198−216

[19] Tu S, Zheng Wenting, Kohler E, et al. Speedy transactions in multicore in-memory databases [C] //Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 18−32

[20] Mao Hongzi, Schwarzkopf M, Venkatakrishnan S B, et al. Learning scheduling algorithms for data processing clusters [C] //Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2019: 270−288

[21] Kristo A, Vaidya K, Çetintemel U, et al. The case for a learned sorting algorithm [C] //Proc of the 2020 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2020: 1001−1016

[22] Marcus R, Negi P, Mao Hongzi, et al. Neo: A learned query optimizer[J]. Proceedings of the VLDB Endowment, 2019, 12(11): 1705−1718

[23] Marcus R, Negi P, Mao Hongzi, et al. Bao: Making learned query optimization practical [C] //Proc of the 2021 Int Conf on Management of Data. New York: ACM, 2021: 1275−1288

[24] Kraska T, Alizadeh M, Beutel A, et al. SageDB: A learned database system [C/OL] //Proc of the 9th Biennial Conf on Innovative Data Systems Research. 2019 [2023-02-27].https://www.cidrdb.org/cidr2019/papers/p117-kraska-cidr19.pdf

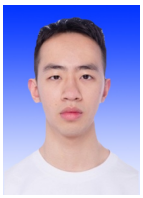
[25] Ding Jialin, Marcus R, Kipf A, et al. SageDB: An instance-optimized data analytics system[J]. Proceedings of the VLDB Endowment, 2022, 15(13): 4062−4078

[26] Li Pengfei, Hua Yu, Jia Jingnan, et al. FINEdex: A fine-grained learned index scheme for scalable and concurrent memory systems[J]. Proceedings of the VLDB Endowment, 2021, 15(2): 321−334

[27] Dai Yifan, Xu Yien, Ganesan A, et al. From WiscKey to Bourbon: A learned index for log-structured merge trees [C] //Proc of the 14th USENIX Conf on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2020: 155−171

[28] Li Pengfei, Hua Yu, Zuo Pengfei, et al. ROLEX: A scalable RDMA-oriented learned key-value store for disaggregated memory systems [C] //Proc of the 21st USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2023: 99−114

[29] Meng Xiaofeng, Ma Chaohong, Yang Chen. Survey on machine learning for database systems[J]. Journal of Computer Research and Development, 2019, 56(9): 1803−1820 (in Chinese)
(孟小峰, 马超红, 杨晨. 机器学习化数据库系统研究综述[J]. 计算机研究与发展, 2019, 56(9): 1803−1820)



Tang Chuzhe, born in 1996. PhD candidate. Student member of CCF. His main research interests include parallel and distributed database systems.

唐楚哲, 1996 年生. 博士研究生. CCF 学生会员. 主要研究方向为并行和分布式数据库系统.

Wang Zhaoguo, born in 1986. PhD. Tenure-track associate professor at the School of Software, Shanghai Jiao Tong University. Member of CCF. His main research interests include the fundamental theory and system building of parallel and distributed database systems.

王肇国, 1986 年生. 博士. 上海交通大学软件学院长聘教轨副教授. CCF 会员. 主要研究方向为并行和分布式数据库系统的基础理论和系统构建.

Chen Haibo, born in 1982. PhD. Distinguished professor at the School of Software, Shanghai Jiao Tong University. IEEE fellow, distinguished member of CCF. His main research interests include operation systems, distributed systems, and systems security.

陈海波, 1982 年生. 博士. 上海交通大学软件学院特聘教授. IEEE fellow, CCF 杰出会员. 主要研究方向为操作系统、分布式系统与系统安全.